Physically interactive tabletop augmented reality using the Kinect

Sam Corbett-Davies*
University of Canterbury
New Zealand

Richard Green†
University of Canterbury
New Zealand

Adrian Clark‡
Human Interface Technology Lab NZ
New Zealand

ABSTRACT

In this paper we present a method for allowing arbitrary objects to interact physically in an augmented reality (AR) environment. A Microsoft Kinect is used to track objects in 6 degrees of freedom, enabling realistic interaction between them and virtual content in an tabletop AR context. We propose a point cloud based method for achieving such interaction. An adaptive per-pixel depth threshold is used to extract foreground objects, which are grouped using connected-component analysis. Objects are tracked with a variant of the Iterative Closest Point algorithm, which uses randomised projective correspondences. Our algorithm tracks objects moving at typical tabletop speeds with median drifts of 8.5 % (rotational) and 4.8 % (translational). The point cloud representation of foreground objects is improved as additional views of the object are visible to the Kinect. Physics-based AR interaction is achieved by fitting a collection of spheres to the point cloud model and passing them to the Bullet physics engine as a physics proxy of the object. Our method is demonstrated in an AR application where the user can interact with a virtual tennis ball, illustrating our proposed method's potential for physics-based AR interaction.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Artificial, augmented, and virtual realities*

Keywords

Augmented reality, interaction, physical simulation

*samcorbettdavies@gmail.com
†richard.green@canterbury.ac.nz
‡adrian.clark@canterbury.ac.nz

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IVCNZ '12, November 26 - 28 2012, Dunedin, New Zealand
Copyright 2012 ACM 978-1-4503-1473-2/12/11 ...$15.00.

1. INTRODUCTION

Augmented reality (AR) is slowly moving from being a simple visualisation tool to an immersive, interactive medium. Interactive augmented reality is an emerging field [2], with applications in entertainment [7, 13, 20, 17, 3, 12, 16], education [3, 8], user interfaces [8, 17, 20, 19] and rehabilitation [14]. Three main AR interaction paradigms currently exist: marker-based, model-based and appearance-based.

1.1 Marker-Based Interaction

The oldest AR interaction technique used specially-designed fiducial markers which were tracked in the real world and used for specifically-defined interactions with AR objects. One of the earliest such methods [12], used markers on "mallets" to allow users to play air hockey in augmented reality. A number of "paddle-based" interaction methods have been developed, including [8], which allowed users to manipulate virtual household objects with a marked "paddle" in an interior design application. A variety of interactions were made possible by executing different gestures with the paddle.

Marker-based interaction techniques are inherently limited in the scope of interactions possible, because every interaction device must be marked. This requirement also limits the possibility of natural interaction, although more recently [19] used a multi-coloured glove to detect hand poses for interaction. Our research aims to develop an interaction method much more flexible than anything possible with marker-based interaction, by allowing arbitrary objects to interact with an AR scene.

1.2 Model-Based Interaction

Model-based interaction techniques fit predefined models to their real-world object counterparts and use these models for interactions with virtual content [18]. These have become particularly popular since the introduction of the Microsoft Kinect, a consumer-priced depth sensing camera, because depth data allows for much more robust model fitting. [16] and [9] are two notable examples of model-based interaction using the Kinect; the former fits a full-body skeleton and the latter an articulated hand model. Both methods allow complicated natural interaction without markers by detecting gestures and poses. They also illustrate the potential for model-based tracking to represent non-rigid objects.

Model-based tracking lends itself to physics simulations, because once the model is oriented to match the real object it can easily be passed to a physics engine. In [17] a physics proxy of a remote-controlled Lego forklift is used for physical interaction between the forklift and virtual crates.

As with marker-based methods, the main disadvantage of model-based interaction is the lack of flexibility, as all objects considered for interaction must have predefined models. Attempts have been made to generalise the models to fit a class of objects [16, 9], but these often encounter problems due to object variations within the class (such as body size and shape for human pose detection). Our proposed method attempts to track objects without the need for predefined models, with the assumption that objects undergo minimal non-rigid deformation.

1.3 Appearance-Based Interaction

Appearance-based tracking techniques are able to track arbitrary objects by automatically detecting features of the object to track [6]. This has lead us to use the term appearance-based interaction to refer to the use of arbitrary objects for interaction. In KinectFusion [7], a moving Kinect is simultaneously localised while a dense 3D reconstruction of the scene is created. Objects in the scene are assumed to be static for localisation, with outlier points treated as foreground objects. The authors demonstrate how the intersections between static and foreground objects can be used for multi-touch interaction in an AR finger painting application. They also show a physics simulation in the reconstructed room, but little physics-based foreground interaction is evident[1].

The focus of KinectFusion was the real-time reconstruction of a scene from depth information, not the facilitation of interactions within that scene. For example, an object moving independently of the camera can cause localisation (and therefore foreground object tracking) to fail. By fixing the Kinect in space we avoid the need for localisation and aim to unambiguously determine the 6 degrees of freedom motion of foreground objects. KinectFusion also requires a powerful GPU and a large amount of graphics memory to achieve its real-time performance. Our aim is to develop a method for real-time AR interaction that does not require such hardware.

Our method is built upon the work presented in [13]. This system used the Kinect to map the environment, but did not track moving objects, so dynamic interaction was not well handled. For example, real objects appeared frictionless when moved underneath AR content. By tracking real-world objects, our system allows friction and collisions to be better modelled.

2. BACKGROUND

The process of aligning point clouds in space (known as *registration*) is well studied, with the most popular algorithm being Iterative Closest Point (ICP) [1, 5]. Generally, this algorithm finds correspondences between points in two point clouds (the "source" and the "target") and then computes the transformation that minimises the error between corresponding points. The transformation is then applied to the source cloud and the process is repeated, with correspondences found using this new transformation, until convergence. ICP is guaranteed to converge to a local minima [1].

The simplest implementation of ICP ("point-to-point") matches each point in the source to its nearest point in the target, and minimises the sum of squared distances between matching points. A number of variants of the algorithm exist, most of which alter how correspondences are determined or what error metric the transformation is calculated to minimise. Rusinkiewicz and Levoy evaluate these varients in [15] and recommend *Projective* ICP for real-time applications using point clouds from a projective device (such as the Kinect). This variant finds correspondences by projecting source points into the target cloud from the Kinect's perspective (see Fig. 1). This means correspondences can be found in constant time, avoiding the expensive process of searching for closest points. This variant requires the use of the point-to-plane error metric (see Fig. 1), which allows points to slide over each other. This ICP variant was used very successfully on point cloud data from the Kinect in [7].

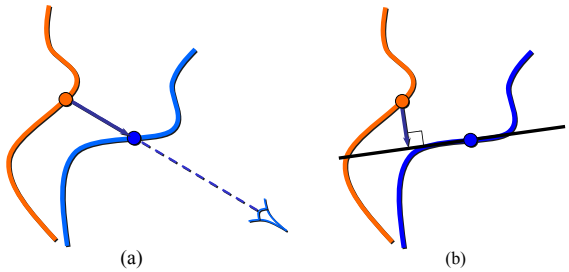


Figure 1: An illustration of Projective Iterative Closest Point. (a) shows how correspondences are found by projecting each source point into the target cloud, while (b) shows the point-to-plane error metric.

3. CONSTRAINTS

The goal of this paper is to demonstrate a framework for arbitrary interaction, so we relaxed most of the constraints placed on the marker and model-based methods discussed earlier. The main constraint is that the Kinect must remain stationary, meaning we do not have to distinguish between camera movement and foreground movement. For simplicity, the application presented in this paper only uses one camera, displaying AR scenes from the perspective of the Kinect. However, typical AR applications allow the user to vary the perspective they observe the scene from, requiring a moving camera. Using the same setup as Piumsomboon *et al.* [13], our proposed method could be easily adapted to achieve this by adding a second camera, whose transformation relative to the Kinect is determined by tracking a printed marker. We also assume objects only undergo rigid transformations, although our algorithm can cope with small deformations of foreground objects.

4. METHOD

This section details the setup and algorithms we use to allow arbitrary objects to interact physically in AR scenes.

4.1 Hardware Setup

The physical setup of our proposed method is shown in Fig. 2. The hardware required is minimal; unlike [19] and [8] we do not require artificial markers on interactive objects, allowing more natural interaction. The Kinect is mounted, facing down, on a stand approximately 1 m above the tabletop environment where interaction is to occur. The right-handed world coordinate system is established with its origin

[1]http://www.youtube.com/watch?v=quGhaggn3cQ

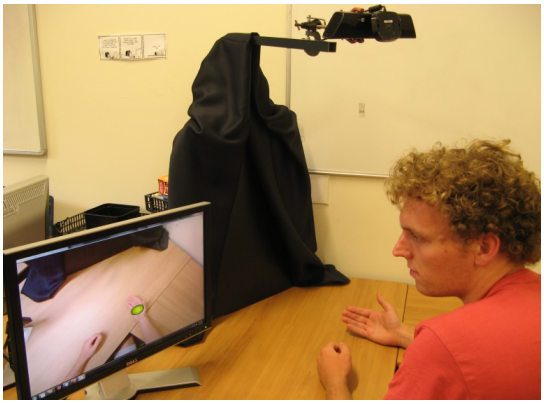


Figure 2: The hardware setup of our proposed method. The only hardware required is a Kinect, mounted at least 80 cm above the tabletop.

at the Kinect's viewpoint and its positive z-axis facing away from the Kinect. We assume gravity acts down the positive z-axis in physical simulations.

4.2 Software

Our proposed method is built on top of four existing open-source software libraries:

1. *Point Cloud Library (PCL)*[2]*:* PCL is an emerging library for working with point cloud data. It has been rapidly developed in response to the huge popularity of the Kinect, but remains experimental and under-documented in some areas. PCL is used for most of the point cloud manipulation in our proposed method, most notably for point cloud registration using their implementation of ICP. PCL includes a grabber framework to retrieve point clouds from the Kinect, which uses OpenNI[3], a low-level library for interfacing to the Kinect.
2. *OpenCV*[4]*:* OpenCV is a popular image processing library, which we use for foreground segmentation and filtering. We also use the cvBlob library[5] for connected-component analysis to group the foreground into objects, as we found was faster than any PCL alternative.
3. *Bullet Physics*[6]*:* Bullet physics is an open-source physics engine, capable of simulating both rigid and soft body dynamics. We use it for simulating physical interaction with AR content, by approximating foreground objects as a collection of spheres.
4. *OpenSceneGraph (OSG)*[7]*:* OSG is a 3D graphics toolkit that optimises rendering using a scene graph data structure. We use it for visualising the AR scene. It also contains a plugin for Bullet Physics, making it easy to render physical simulations.

4.3 Point Cloud Downsampling

The Kinect outputs a 640 by 480 RGB-depth image, which PCL turns into a 3D point cloud. At this resolution there are over 300,000 points in the cloud, which increases processing times beyond that of real-time performance. For this reason our pipeline begins by downsampling the point cloud at increments of 5 in each dimension without smoothing. Our experiments have shown that reducing the resolution of the point cloud by a factor of 25 does not noticeably degrade the tracking performance. In fact, tracking is improved because each frame can be processed faster, decreasing the possible between-frame pose variation of the tracked objects.

[2]http://pointclouds.org
[3]http://www.openni.org
[4]http://opencv.willowgarage.com
[5]http://code.google.com/p/cvblob
[6]http://bulletphysics.org
[7]http://www.openscenegraph.org

Algorithm 1 Foreground segmentation using per-pixel depth comparison.

```
1:  D ← current depth map
2:  B ← background depth map
3:  F ← foreground map
4:  for each pixel u in depth map D do
5:      d ← D(u)
6:      b ← B(u)
7:      if ∃ a pixel x touching u: (d − D(x)) > ε2) then
8:          F(u) ← false
9:      else if ε1 b > d then
10:         F(u) ← true
11:     else
12:         F(u) ← false
13:         if b < d then
14:             B(u) ← d
```

4.4 Foreground Segmentation

We then have to determine which parts of the scene should be considered for interaction. Algorithm 1 describes the foreground segmentation algorithm, which works on the assumption that the that furthest depth recorded at each pixel is the distance to the background at that pixel. Therefore any pixel with a depth smaller than some proportion (ϵ_1) of the background depth at that pixel is considered part of the foreground. Through experimentation, we determined that the best value for ϵ_1 is 0.985, which minimises false-positive foreground pixels due to sensor noise while still segmenting out a hand placed flat on the tabletop.

Foreground objects are found by performing connected-component analysis on the foreground map using the algorithm described by Chang *et al.* in [4]. Only components with over 50 points are considered to be objects. Notice the algorithm considers any point that has an adjacent point that is a given threshold (ϵ_2) closer to the Kinect than it is as part of the background (line 7). This is so that overlapping objects at different depths are not connected during connected-component analysis. We use an ϵ_2 value of 3 cm. The result is a segmented, labeled map of all foreground objects.

4.5 Object Tracking using ICP

Identified objects are tracked in 6-degrees of freedom using the Iterative Closest Point algorithm. Although Projective ICP is recommended by [15] and [7], the former assumed normals were available for each point in the target cloud, while the latter used the GPU to rapidly calculate these. Without normals the point-to-plane metric cannot be used, and Projective ICP does not converge. We found that even with normals Projective ICP often did not converge to a reasonable solution. The problem of minimising the sum of squared point-to-plane errors only has a closed form solution

(a) (b)

Figure 3: A visualisation of the point cloud representation of foreground objects. (a) shows two hands being tracked using real Kinect data, while (b) shows the synthesised data used to test the tracking algorithm.

if small transformations are assumed [11]. Projective ICP must therefore run at very high frame rates to ensure only small transformations occur frame-to-frame. PCL took up to 15 ms to estimate normals for the downsampled Kinect point cloud, too slow to make Projective ICP viable. Point-to-point ICP was also too slow, as a kd-tree had to be built and searched to find neighbouring points to match.

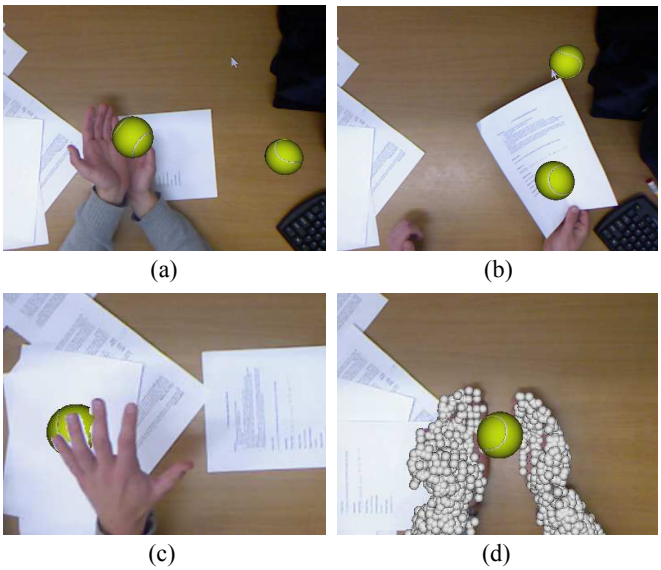
To solve this problem we used a slightly modified version of the projective ICP algorithm. Each source point was matched to a randomly selected target point within a 5-by-5 window of its projected match. Matches were rejected if the point-to-point distance was greater than 5 mm. We found that, on aggregate, this method selects better matches than Projective ICP, while being just as fast. At each ICP iteration, the transformation that minimises the sum of squared point-to-point distances is found using singular value decomposition.

4.6 Model Updating

An internal point-cloud model of each object is kept by the system, which become the "source" clouds for ICP. This model is constantly updated with new information from the Kinect, both to improve the quality of the tracking and to allow for minor non-rigid deformation of foreground objects. Model points are assigned a weight between -1 and 1. This weighting reflects our confidence that the model point is indeed part of the object. Points with a positive weighting are considered part of the object, while points with a weight less than -1 are removed entirely. A point's weight is increased when it's position matches the Kinect's observation, and decreased when it does not. The updating process is as follows:

1. Project all model points into the coordinate space of the Kinect's depth map. The Kinect's projection matrix is estimated when the application starts using PCL's *OrganizedNeighbor* class.
2. For each pixel in the depth map, record the model point ($\mathbf{m}$) closest to the camera that projects into that pixel. The object containing this point records the foreground component at this pixel. All other model points that project into that pixel are not visible to the Kinect so are left unchanged by the update process.
3. Look up the corresponding point ($\mathbf{p}$) in the Kinect's observed point cloud. If $\|\mathbf{p}-\mathbf{m}\| < \epsilon_3$ the point is an inlier, and the connected-component label of the foreground point recorded. The model point's weighting is increased by α and its position is set to $\beta\mathbf{p}+(1-\beta)\mathbf{m}$.
4. If $\|\mathbf{p}\| - \|\mathbf{m}\| > \epsilon_3$ the model point lies significantly in front of the observation, decrease the model point's weighting by α, as the model point does not match the observation.
5. If $\epsilon_3 < \|\mathbf{m}\| - \|\mathbf{p}\| < \epsilon_4$ the observation lies in front of the model point, add the observed point to the model with a weighting of -1, leave the model point unchanged.
6. If $\|\mathbf{m}\| - \|\mathbf{p}\| > \epsilon_4$ the observation lies far in front of the model point, leave the model point unchanged.
7. For all foreground components that have at least twice as many points in this model than any other (as found in step 2), add the remainder of their points to the model with a weighting of -1.

After significant experimentation the following threshold and constant values were determined to be best: $\epsilon_3 = 1$ cm, $\epsilon_4 = 3$ cm, $\alpha = 0.5$ and $\beta = 0.8$.

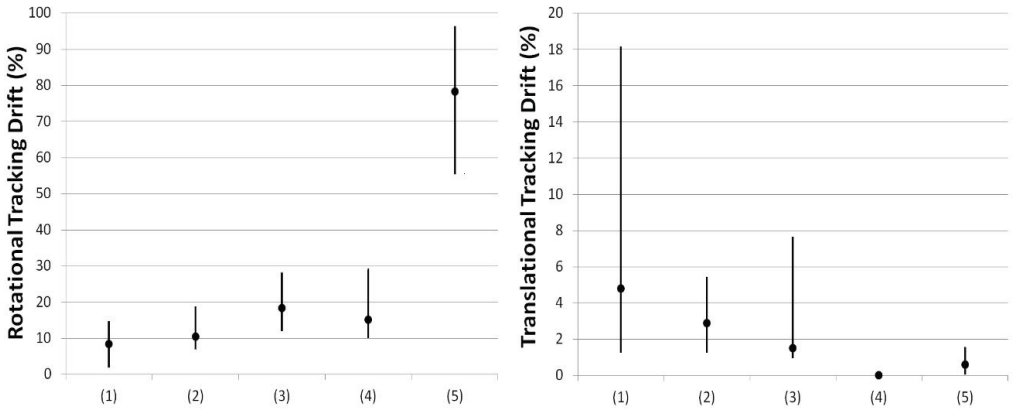


(a) (b) (c) (d)

Figure 4: Screenshots of our demonstration AR application. (a) and (b) show the user interacting with virtual tennis balls. (c) shows a tennis ball being occluded by the user's hand using our custom fragment shader, while (d) shows the spheres used as physics proxies for foreground objects.

4.7 Physics Simulation

Physical proxies of real objects in the scene are created in the Bullet physics engine to facilitate physics simulation. The tabletop is represented with a triangulated mesh, with each vertex set to the corresponding value in the background depth map. Foreground objects are represented with collections of spheres with 5 mm radii. This allows complex real-world objects (such as a hand) to be represented by a collection of simple physics objects (spheres). Only a subset of model points are represented; a 3D grid (with cube width 15 mm) is fit over the model point cloud and a single sphere used for each occupied box in the grid. The motion of these spheres is determined by the most recent transformation the object has undergone, as found in the ICP step. Bullet handles collisions between these spheres and other



Speed	(1)	(2)	(3)	(4)	(5)
Rotational (rad s^{-1})	1	2	1	3	0.01
Translational (cm s^{-1})	5	5	10	0	30

Figure 5: Results of the tracking algorithm tests, showing rotational and translational drift under various testing speeds. The rotational and translational speeds that each test was conducted at are shown in the table. The median drift of five trials is shown, along with maximum and minimum values.

virtual content, allowing the user to dynamically interact with virtual objects, pushing them, carrying them and even hitting them with real-world objects. To minimise the computational load, our simple demonstration application only included tennis balls for the user to play with, as seen in Fig. 4.

4.8 Rendering

The rendering done by OpenSceneGraph in our application is very simple, with the Kinect's RGB view of the scene drawn as the background and virtual content rendered on top. One significant improvement in our proposed method compared to typical AR applications is the ability to perform realistic occlusion of AR content. The Kinect's depth map allows us to determine on a per-pixel basis which areas of virtual objects should not be drawn, as they lie behind real-world objects. We wrote a custom fragment shader for this purpose, which discards fragments with a greater distance from the Kinect than the real-world object at the same pixel. The 32-bit floating-point depth values are passed to this shader in a texture, packed into four 8-bit colour channels (RGBA). This allows the full depth resolution to be recovered in the fragment shader, which typical only allow 8-bit colour resolution. Figure 4 shows a virtual ball being occluded by a real hand.

5. RESULTS

Existing literature on interactive augmented reality rarely includes quantitative results, so comparisons between our proposed method and existing work are difficult to make. We evaluated the accuracy of our tracking algorithm on synthesised Kinect point clouds, tracking the known motion of a 3D model. This method avoids the challenge of determining the ground truth 6DOF motion of an object. The point cloud was generated by projecting the points of 3D models into the Kinect's image plane, taking the closest point to the Kinect at each image point.Gaussian noise was added to the z value of each point to simulate noise in the Kinect's depth measurements as found in [10].

The simulated test scene (shown in Fig. 3) consisted of a flat background with the a model of the Stanford Bunny[8] placed in front of it to be tracked. In every test the bunny object was translated and rotated in a random but known fashion at various linear and angular speeds. The synthesised Kinect data was updated every frame and passed to our tracking algorithm. The rotational tracking error was determined by finding the angle-axis representation of the rotation between the measured and ground truth transformations and recording the angle. The translational tracking error was found by measuring the distance between the model centroid and the actual centroid of the bunny. Tracking drift is defined as average error as a percentage of distance (either angular or linear) traveled, and is shown in Fig. 5. In the future we will more thoroughly evaluate our tracking framework, particularly the effect of downsampling the point cloud to different resolutions.

Figure 5 shows that at typical tabletop speeds ($5\,\mathrm{cms}^{-1}$ and $1\,\mathrm{rads}^{-1}$) the tracking algorithm drifts by only 8.5 % rotationally and 4.8 % translationally. It also illustrates a problem with the algorithm under fast translation; the model updating step add new points to ensure the centroid moves as expected, but these points are not matched for ICP, so an accurate rotation cannot be found. The maximum drifts (and resulting high variance) seen in Fig. 5 are caused by tracking failure, however this failure is not catastrophic because the model is quickly updated to again reflect the observation.

6. PERFORMANCE AND FUTURE WORK

Despite the optimisations discussed above, our demonstration application was only able to achieve an average frame rate of 12 Hz on an 2.4 GHz Intel quad-core processor. This was fast enough to achieve our goal of realistic AR interaction, but certainly not fast enough for an optimal user experience. Although running on a quad-core processor, our method is essentially single-threaded, and future development will have to take advantage of parallel process-

[8]http://graphics.stanford.edu/data/3Dscanrep/

ing to achieve truly real-time rates. PCL has an emerging GPU library that harnesses the power of Nvidia GPUs using CUDA[9], the same technology that allowed KinectFusion to achieve a frame rate of 30 Hz doing much more complicated tracking and reconstruction [7]. During development of our method we investigated this library, but found it is currently too unstable and lacking some important features. The majority of the algorithms we used are readily parallelisable, so future work will involve porting them to CUDA to run on the GPU.

7. DISCUSSION AND CONCLUSION

This paper has presented a detailed discussion of the development of our interactive augmented reality framework. We have developed a method that allows a user to interact in a physically realistic way in an augmented reality scene using arbitrary objects. Our proposed method facilitates natural interaction, as it does not require markers or artificial constraints of any kind on objects used for interaction. This does, however, limit the complexity of interactions possible. For example, the user cannot "pull" AR objects, only push or carry them. Enabling more complicated natural interaction will require detection of higher-level gestures, such as that done in [19]. We believe the work we have presented could be used to improve the user's experience and level of immersion in augmented reality applications in education and entertainment.

8. REFERENCES

[1] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, Feb. 1992.

[2] M. Billinghurst. The Future of Augmented Reality in Our Everyday Life. In *Proceedings of the 19th International Display Workshops*, Nagoya, Japan, December 2011.

[3] P. Buchanan, H. Seichter, M. Billinghurst, and R. Grasset. Augmented reality and rigid body simulation for edutainment: the interesting mechanism - an AR puzzle to teach Newton physics. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, ACE '08, pages 17–20, New York, NY, USA, 2008. ACM.

[4] F. Chang, C.-J. Chen, and C.-J. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Comput. Vis. Image Underst.*, 93(2):206–220, Feb. 2004.

[5] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2724 –2729 vol.3, apr 1991.

[6] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(5):564 – 577, may 2003.

[7] S. Izadi, R. A. Newcombe, D. Kim, O. Hilliges, D. Molyneaux, S. Hodges, P. Kohli, J. Shotton, A. J. Davison, and A. Fitzgibbon. KinectFusion: real-time dynamic 3D surface reconstruction and interaction. In *ACM SIGGRAPH 2011 Talks*, SIGGRAPH '11, pages 23:1–23:1, New York, NY, USA, 2011. ACM.

[8] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana. Virtual object manipulation on a table-top AR environment. *Proceedings IEEE and ACM International Symposium on Augmented Reality ISAR 2000*, pages 111–119, 2000.

[9] C. Keskin, F. Kirac, Y. Kara, and L. Akarun. Real time hand pose estimation using depth sensors. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1228 –1234, Nov. 2011.

[10] K. Khoshelham and S. O. Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[11] K.-L. Low. Linear least-squares optimization for point-to-plane icp surface registration. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill.

[12] T. Ohshima, K. Satoh, H. Yamamoto, and H. Tamura. AR^2 Hockey: A Case Study of Collaborative Augmented Reality. In *Proc. IEEE VRAIS '98*, pages 268–275, 1998.

[13] T. Piumsomboon, A. Clark, and M. Billinghurst. Physically-based Interaction for Tabletop Augmented Reality Using a Depth-sensing Camera for Environment Mapping. In *Proc. Image and Vision Computing New Zealand (IVCNZ-2011)*, pages 161–166, Dec 2011.

[14] A. Rizzo and G. J. Kim. A SWOT analysis of the field of virtual reality rehabilitation and therapy. *Presence: Teleoper. Virtual Environ.*, 14(2):119–146, Apr. 2005.

[15] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

[16] J. Shotton, A. W. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, pages 1297–1304. IEEE, 2011.

[17] P. Song, H. Yu, and S. Winkler. Vision-based 3D finger interactions for mixed reality games with physics simulation. In *Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, VRCAI '08, pages 7:1–7:6, New York, NY, USA, 2008. ACM.

[18] H. Uchiyama and E. Marchand. Object detection and pose tracking for augmented reality: Recent approaches. In *18th Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV)*, Feb. 2012.

[19] R. Y. Wang and J. Popović. Real-time hand-tracking with a color glove. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 63:1–63:8, New York, NY, USA, 2009. ACM.

[20] A. D. Wilson. Depth-Sensing Video Cameras for 3D Tangible Tabletop Interaction. *Second Annual IEEE International Workshop on Horizontal Interactive HumanComputer Systems TABLETOP07*, 106(5):201–204, 2007.

[9] http://www.nvidia.com/object/cuda_home_new.html